Integration von SDN in eine virtualisierte IT-Topologie

Jonas Sell[1] · Evren Eren[2]

[1]FH Dortmund
jonas.sell@fh-dortmund.de

[2]HS Bremen
Evren.Eren@hs-bremen.de

Zusammenfassung

Software-defined Networking (SDN) wird immer relevanter in aktueller Netzwerktechnik. Große, komplexe, sich schnell verändernde Netze sollen durch diese Technologie einfacher verwaltet werden. Die Umstellung von herkömmlichen Netzwerken auf SDN ist eine Herausforderung und die bei aktuellen SDN-Controllern verfügbaren Schnittstellen bieten sich für eine Verknüpfung mit bestehenden Systemen an. In diesem Paper wird gezeigt, wie eine vorhandene IT-Topologie auf SDN umgestellt und mit bestehenden Komponenten verbunden wird, um den Funktionsumfang des Netzwerkes für Intrusion Detection zu erhöhen. Des Weiteren wird eine eigens entwickelte Anwendung vorgestellt, die Snort und OpenDaylight verknüpft. Sie nimmt Alerts von Snort via Unix-Socket entgegen und generiert auf Basis dessen einen Flow für einen OpenDayLight-basierten SDN-Controller zur Kontrolle der Kommunikation.

1 Einleitung

Mit SDN ist ein neuer Standard geschaffen worden um den immer mehr zunehmenden Verwaltungsaufwand von komplexen Netzwerken zu reduzieren: Viele verschiedene Systeme verschiedener Hersteller mit unterschiedlichen Schnittstellen und Administrationswerkzeugen erhöhen die Administration enorm. SDN will diese Probleme lösen indem eine Lösung geschaffen wird, die die zentrale, standardisierte und damit produktübergreifende Verwaltung erlaubt [FeRZ13].

Um das SDN-Konzept umsetzen zu können muss das Netzwerk aus mindestens einem SDN-fähigen Gerät und einem Controller bestehen. Der Controller ist die zentrale Verwaltungseinheit des Netzwerkes und kennt dessen Struktur und Konfiguration: Ein direkter Zugriff auf das Netzwerkgerät z.B. per Konfigurationsoberfläche ist damit nicht mehr nötig. Durch die Zentralisierung wird die Verwaltung deutlich vereinfacht und auf Änderungen in der Netzstruktur kann schneller und flexibler reagiert werden. SDN ermöglicht eine Abstraktion des Network-OS, da es die Steuerungs- und Datenebene (Control Plane und Data Plane) voneinander trennt, so dass logische Netzdienste auf heterogener physikalischer Infrastruktur operieren können und dadurch flachere Netze resultieren [Ere15]. Dabei wird die IT-Infrastruktur von einem SDN-Controller verwaltet und gesteuert, der logisch und physikalisch unabhängig von der Infrastruktur agiert.

P. Schartner · A. Baumann (Hrsg.) · D•A•CH Security 2017 · syssec (2017) 421-430.

Für die Kommunikation zwischen Controller und SDN-Gerät hat sich OpenFlow als De-Facto Standard entwickelt. Die Open Networking Foundation [Fou15a] widmet sich der Verbreitung und Implementierung von SDN und OpenFlow. Mit OpenFlow wird eine standardisierte Schnittstelle geschaffen um direkten Zugriff auf die Forwarding-Schicht eines SDN-Gerätes zu ermöglichen. Dies wird sowohl physikalisch als auch virtuell auf Basis eines Hypervisors ermöglicht. Der gesamte Steuerungsfluss aller Netzwerkgeräte kann somit an einer zentralisierten Stelle über Regeln, sogenannter Flows, erfolgen.

Insbesondere durch Virtualisierung und IT-Trends wie Microservices und containerbasiertes Deployment ist es essentiell Netzwerke an dynamische Lastsituationen anpassen zu können. Als Beispiel können bei steigenden Nutzerzahlen automatisch zusätzliche Container mit der benötigten Anwendungssoftware hochgefahren und mit in den Clusterverbund aufgenommen werden. Hierzu ist es notwendig nötige Routen, Firewall-Freischaltungen und Loadbalancer-Konfigurationen automatisiert vorzunehmen. Mit OpenFlow ist dies ohne Hardwareänderungen automatisiert und zentral möglich [Fou12]. Erst in den letzten Jahren ist das SDN-Konzept so weit entwickelt worden, dass es in Produktivumgebungen nutzbar ist. Damit bietet es sich an, es in bestehende Architekturen zu integrieren und mit den dort vorhandenen Komponenten zu verknüpfen:

In diesem Artikel wird eine Beispielimplementierung vorgestellt, bei welcher eine virtualisierte IT-Sicherheitstopologie um SDN erweitert und Sicherheitsmechanismen wie beispielsweise Intrusion Detection mit SDN verknüpft wurde um Angriffe automatisch direkt am SDN-Switch abwehren zu können.

2 Analyse bestehender Systeme der IT-Topologie

Das auf SDN zu portierende Netzwerk besteht aus einem Gateway, welches den Zugang zum öffentlichen Netz bereitstellt. Daran angeschlossen ist eine VPN-Einwahl-DMZ, eine Demilitarisierte Zone und ein in VLANs unterteiltes Intranet, welches diverse Dienste bereitstellt. Eine Übersicht über den relevanten Teil des Netzwerkes zeigt Abbildung 1.

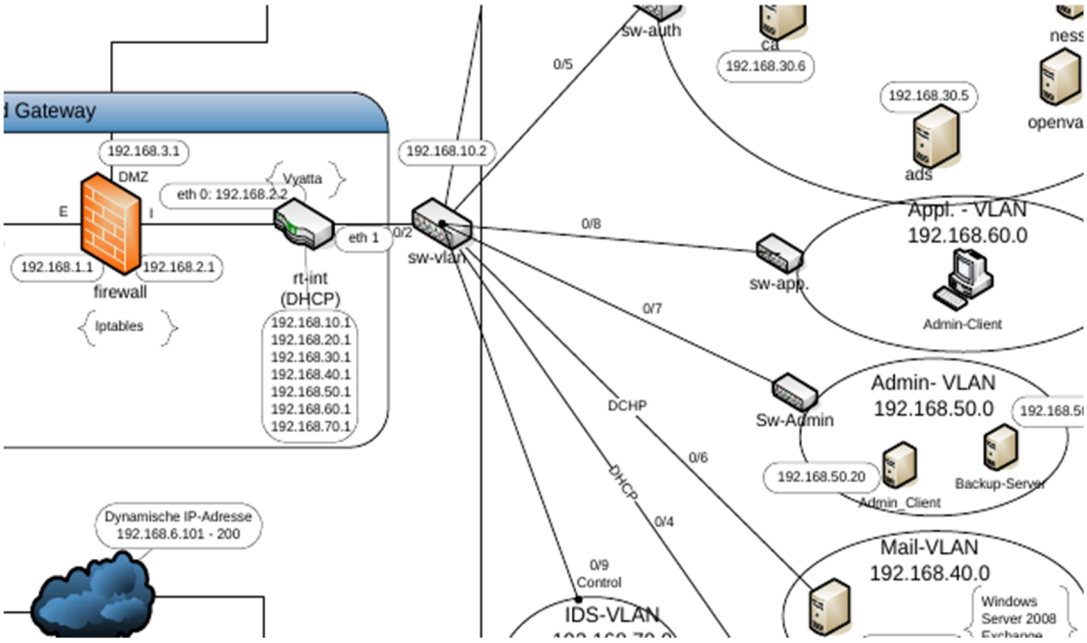

Abb. 1: Ausschnitt des Netzplans des Versuchsnetzwerks

Das komplette Netzwerk ist mittels der Virtualisierungssoftware QEMU[1] und einem virtuellen Netzwerk – realisiert durch Virtual Distributed Ethernet (VDE)[2] – umgesetzt. In dieser Konfiguration ist die Nutzung von SDN respektive OpenFlow nicht möglich, da VDE SDN-Technologien nicht unterstützt. Der Hypervisor QEMU muss aus den Source-Code-Quellen selbstkompiliert werden um VDE nutzen zu können, da die Installation aus den Paket-Repositorys des Betriebssystems keine Unterstützung für VDE mit sich bringt. Snort[3] bildet die Grundlage für das Intrusion Detection System, welches Angriffe auf das Intranet erkennen soll. Das IDS befindet sich im Intranet in einem eigenen VLAN. Die vorhandene Installation ist auf dem Stand von Ende 2010 und damit stark veraltet. Alle virtuellen Maschinen des Intranets sind an einem Switch, an welchem das 802.1Q-Tagging für die Realisierung der VLANs geschieht, angeschlossen. Die Steuerung des Systems erfolgt über zwei Scripts. Das erste Script steuert das Netzwerk: Switches werden erstellt und untereinander verknüpft. Das zweite Script ermöglicht die Verwaltung und Steuerung der virtuellen Maschinen. Beide Scripts sind direkt für den Einsatz mit QEMU und VDE konzipiert.

3 Auswahl der zu verwendenden Technologien

Für die Integration von SDN und OpenFlow in das Referenzsystem mussten vorhandene Technologien ersetzt und ergänzt werden. Der Fokus sollte dabei auf der Nutzung von OpenSource-Software liegen.

Virtualisierung

Um OpenFlow nutzen zu können mussten die virtuellen Switches diese Technologie unterstützten, was durch VDE nicht gegeben ist. Als Alternative bietet sich Open vSwitch[4] an. Die Software kann sowohl mit OpenFlow als auch mit gängigen Virtualisierungstechnologien genutzt werden. Mit dem Austausch von VDE durch Open vSwitch musste auch eine Alternative zur Verwaltung der virtuellen Maschinen gefunden werden, da das ursprünglich verwendete Steuerungsscript auf VDE zurückgreift. Daher sollte die Steuerung durch das Programm libvirt[5] beziehungsweise virsh[6] ersetzt werden. Libvirt ist eine Bibliothek, welche diverse Funktionen zur Nutzung von Virtualisierung bereitstellt. Die Steuerung erfolgt dabei über das Kommandozeilenprogramm virsh. Libvirt unterstützt dabei auch die Nutzung von Open vSwitch, weshalb sich die Software als Ersatz für die bisherige Verwaltung der virtuellen Maschinen anbietet.

Controllersoftware

Für die Steuerung der SDN-Switches mittels OpenFlow wird eine Controller-Software benötigt. Damit diese mit anderen Komponenten verknüpft werden kann, müssen Schnittstellen nach außen verfügbar sein. In die nähere Auswahl kam OpenDaylight[7] da die Anforderung einer Schnittstelle durch eine REST-API erfüllt ist und das Projekt aktiv unterstützt wird.

---

1 http://www.qemu-project.org

2 http://vde.sourceforge.net

3 http://snort.org

4 http://openvswitch.org

5 http://libvirt.org

6 https://libvirt.org/virshcmdref.html

7 http://opendaylight.org

Intrusion Detection System

Das schon vorhandene Intrusion Detection System auf Basis von Snort ist stark veraltet und musste durch eine neuere Version ersetzt werden. Mit einer aktuellen Snort-Installation stehen weiterhin zusätzliche Schnittstellen zur Entgegennahme von Alarmen zur Verfügung wie Unix-Domain-Sockets.

Schnittstelle zwischen IDS und Controllersoftware

Da eine direkte Verknüpfung zwischen Snort und OpenDaylight nicht möglich ist, ist ein zusätzlicher Zwischenschritt in Form eines Programms nötig. Die Aufgabe des Programms ist es, von Snort generierte Alerts über einen Unix-Domain-Socket entgegen zu nehmen. Auf Basis der Alerts werden Regeln in Form von Flows erstellt und an OpenDaylight via REST-API gesendet. Die generierten Flows müssen dazu im XML-Format vorliegen und sollen die Kommunikation zwischen angreifendem und angegriffenem Host unterbrechen. Als Programmiersprache soll Python genutzt werden, da damit die Nutzung von Domain-Sockets, der REST-API und die Erstellung von XML-Dokumenten möglich ist. Prinzipiell könnte hier jede andere Sprache zum Einsatz kommen, welche diese Anforderungen unterstützt.

Einen Überblick über die verwendeten Technologien gibt Tabelle 1.

Tab. 1: Übersicht über zu verwendenden Technologien

Technologie	Zweck	Grund
Open vSwitch	Ersatz für VDE	OpenFlow-Unterstützung
libvirt	Virtualisierung	Unterstützung von Open vSwitch
OpenDaylight	OpenFlow Controller	aktuelles Projekt, Unterstützung der Anforderungen
Python	Umsetzung des Programms	Programmiersprache entspricht den Anforderungen
Snort	IDS	Einsatz auch in der ursprünglichen Umgebung

4 Umsetzung

Die Integration von SDN in eine bestehende Versuchsplattform bestand aus fünf Punkten: Anpassung der Entwicklungsumgebung (des Virtualisierungshosts) zur Nutzung von libvirt und Open vSwitch, Installation des SDN-Controllers, Anpassung des Intrusion Detection Systems und die Erstellung des Programms als Schnittstelle zwischen Snort und OpenFlow.

4.1 Anpassung des Virtualisierungshosts

Die bestehenden virtuellen Maschinen des Netzwerkes mussten in die libvirt-Umgebung übertragen werden. Dazu genügte es, jede VM über libvirt neu anzulegen und das schon existierende Festplattenimage als Grundlage zu nutzen. Nur die Vergabe der MAC-Adressen der Netzwerkinterfaces bedurfte Beachtung, da die ursprünglichen Adressen beibehalten werden mussten. Wird dieser Punkt nicht beachtet kann es zu ungewollten Nebeneffekten kommen, da unter Umständen neue Netzwerkinterfaces mit verändertem Namen innerhalb der VM angelegt werden und dadurch Dienste nicht mehr korrekt arbeiten. Damit die virtuellen Netzwerkkarten mit

Open vSwitch verbunden werden konnten, musste die Konfiguration der VM manuell geändert werden, indem folgende Zeilen hinzugefügt bzw. angepasst wurden:

```
<interface type='bridge'>
<source bridge='$NamederBridge'/>
<virtualport type='openvswitch'/>
```

Zusätzlich wurde im Intranet im passenden VLAN eine neue virtuelle Maschine für den SDN-Controller OpenDaylight erzeugt. Die neue VM hatte zwei Netzwerkinterfaces: Ein Interface bildete die Schnittstelle in das Testnetzwerk, die andere Schnittstelle ist vom Virtualisierungshost erreichbar. Ohne diese zweite Schnittstelle konnte später der Switch nicht mit dem Controller verbunden werden, da der Virtualisierungshost den internen IP-Adressraum des Intranets des Versuchsnetzes nicht kannte und somit nicht erreichen konnte. Der Netzwerkanschluss zum Hostsystem wurde über eine simple Linux-Bridge als virtuelles Netzwerkinterface realisiert.

Damit Snort später Alerts über einen Unix-Socket ausgab, musste in der Konfiguration die Option output alert_unixsock aktiviert werden. Nach einem Neustart nutze Snort einen unter /var/log/snort/snort_alert vorhandenen Socket, um Alerts auszugeben. Der Socket wird nicht von Snort erstellt, sondern muss anders erzeugt werden.

4.2 Anpassung des Netzwerkes

Die vorhandenen VDE-Switches wurden durch Open vSwitch ersetzt. Das Tool ovs-vsctl $Switchname erstellt einen neuen Switch auf Basis von Open vSwitch. Wurde jeder VDE-Switch durch einen Open vSwitch-Switch ersetzt und die VMs zu deren Nutzung konfiguriert, konnten diese gestartet werden. Eine Kommunikation zwischen den VMs untereinander war noch nicht möglich, da noch kein 802.1Q-Tagging erfolgte. Dies konnte entweder, wie ursprünglich unter VDE auch, direkt am Switch mittels OpenFlow oder in den virtuellen Maschinen selbst erfolgen. Zunächst wurde der Einfachheit halber das VLAN-Tagging in den VMs realisiert.

Damit der Switch, an welchem später Angriffe abgewehrt werden sollen, mit OpenFlow konfigurierbar ist, musste die Verbindung zwischen Switch und Controller hergestellt werden. Dies erfolgte für Open vSwitch über den Befehl ovs-vsctl set-controller $NamedesSwitches tcp:$AdressedesControllers:6633. Wurde die Verbindung erfolgreich hergestellt, konnte der Switch Flows entgegennehmen. Damit die Grundkommunikation im Versuchsnetz möglich war, mussten mindestens drei Flows existieren. Zwei Flows mussten den Transport von ARP-Paketen abdecken, der dritte Flow regelte den Transport von TCP-Paketen. Zunächst musste ein Flow formuliert werden, welcher ARP-Broadcast-Pakete an alle Ports des Switches weiterleitet. Ein für den Controller OpenDaylight formulierter Flow in Form eines XML-Dokumentes sah wie folgt aussehen:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
   <flow-name>arp2broadcast</flow-name>
   <table_id>0</table_id>
   <id>0</id>
   <priority>100</priority>
   <instructions>
      <instruction>
         <order>0</order>
         <apply-actions>
            <action>
               <order>1</order>
               <output-action>
                  <output-node-connector>ALL</output-node-connector>
                  <max-length>200</max-length>
               </output-action>
            </action>
         </apply-actions>
      </instruction>
   </instructions>
   <match>
      <ethernet-match>
         <ethernet-type>
            <type>2054</type>
         </ethernet-type>
         <ethernet-destination>
            <address>ff:ff:ff:ff:ff:ff</address>
         </ethernet-destination>
      </ethernet-match>
   </match>
</flow>
```

Der Flow matchte auf den Ethernet-Type 2054 (entspricht ARP-Paketen nach IEEE 802[8]) mit dem Ethernet-Ziel ff:ff:ff:ff:ff:ff (MAC der Broadcastadresse). Als Aktion ist die Ausgabe an alle Ports des Switches definiert. Dies bedeutet, dass ARP-Pakete mit dem Ziel der Broadcast-Adresse über alle Ports des Switches weitergeleitet werden. Ähnliche Regeln mussten pro VM für den Transport von ARP- und TCP-Paketen existieren. Diese variierten in der Angabe der Ziel-MAC-Adresse und dem Ausgabeport. Waren diese Flows formuliert und auf dem Switch installiert, war eine Kommunikation zwischen den VLANs untereinander und in das restliche Versuchsnetz möglich.

Damit Snort Traffic überwachen konnte, musste Open vSwitch für Traffic-Duplizierung zur Snort-VM konfiguriert sein. Dies geschah über den Befehl ovs-vsctl -- set Bridge sw-vlan mirrors=@m -- --id=@monitor get Port vnet13 -- --id=@ids get Port vnet12 -- --id=@m create Mirror name=snort select-dst-port=@monitor select-src-port=@monitor output-port=@ids

[8] vgl https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml

4.3 Verknüpfung von Intrusion Detection und SDN

Die Verknüpfung von Snort und OpenDaylight erfolgte über ein Programm, welches Alerts von Snort über einen Unix-Socket entgegennimmt und auf Basis dieser Daten einen SDN-Flow generiert, welcher die Kommunikation zwischen angreifendem und angegriffenem Host unterbindet. Dadurch ergaben sich folgende Anforderungen an das Programm:

- Erstellen des Sockets für Snort
- Verfügbar machen (Anpassung von Zugriffsrechten) des Sockets für Snort
- Auslesen des Snort-Sockets
- Erkennen und entgegennehmen von Alerts über den Socket
- Entpacken bzw. auslesen der im Alert enthaltenen Daten wie Quell-IP, Ziel-IP etc.
- Generierung eines Flows als XML-Dokument auf Basis der ausgelesenen Daten
- Senden des Flows an OpenDaylight über die REST-API
- Löschen von generierten Flows, welche eine vorgegebene Zeit lang aktiv sind, um die Blockade wieder aufzuheben

Für die Umsetzung wurden Bibliotheken genutzt, welche zum Teil über den Python-Paketmanager „pip“ nachinstalliert werden mussten. Eine Übersicht zeigt Tabelle 2:

Tab. 2: Übersicht über genutzte Python-Bibliotheken

Bibliothek	Funktion/Nutzung
socket	benötigt, um UNIX-Domain-Sockets erstellen zu können
os	benötigt für Dateisystem-Funktionen (chown)
alert	Bibliothek zum Entpacken des Snort-Alerts (Originalautor: Nippon Telegraph and Telephone Corporation, 2013)
dpkt	Entpacken der übergebenden Ethernet-Frames für Zugriff auf MAC-/IP-Adressen
time/datetime	zum Auslesen der aktuellen Zeit/des aktuellen Datums
requests	Generierung von HTML-Requests (PUT/DELETE) an den Controller
xml.etree.ElementTree	Erstellung der XML-Dokumente
xml.dom	Erstellung der XML-Dokumente
subprocess	Aufrufen von Systembefehlen, hier systemctl
thread	Erstellung von Threads
snortunsock	Entpacken des Snort-Alerts

Nach Programmstart musste zunächst ein Unix-Domain-Socket unter /var/log/snort/snort_alert angelegt, Zugriffsrechte angepasst und der Snort-Daemon neu gestartet werden, damit Snort den Socket nutzte. Anschließend wurde gewartet bis Alerts über den Socket durch Snort ausgegeben werden. Ein eingehender Alert wurde zunächst entpackt und die darin enthaltenen Da-

ten ausgelesen. Für die Generierung der Flows waren die Angabe der Ziel- und Quell-IP wichtig. Insgesamt wurden zwei Flows erstellt und an den Controller gesendet: Der erste Flow unterbindet die Kommunikation zwischen Ziel und Quelle, der zweite zwischen Quelle und Ziel. Dadurch war sichergestellt, dass auch verbindungslose Netzwerkprotokolle wir UDP bei einem Angriff blockiert werden. Anschließend wurde wieder der Socket auf neue Alerts ausgelesen. Ein erstellter Flow erhielt eine eindeutige ID und wurde in einer Liste abgelegt. Diese Liste wurde regelmäßig auf abgelaufene Einträge durchsucht und entsprechende Flows gelöscht. Da das Programm normalerweise erst weiterläuft, wenn ein Alert über den Socket eintraf, wurde dieser Prozess in einen eigenen Thread ausgelagert. Dadurch war sichergestellt, dass die Liste unabhängig von anderen Ereignissen durchsucht werden kann. Den schematischen Programmablauf zeigt Abbildung 2:

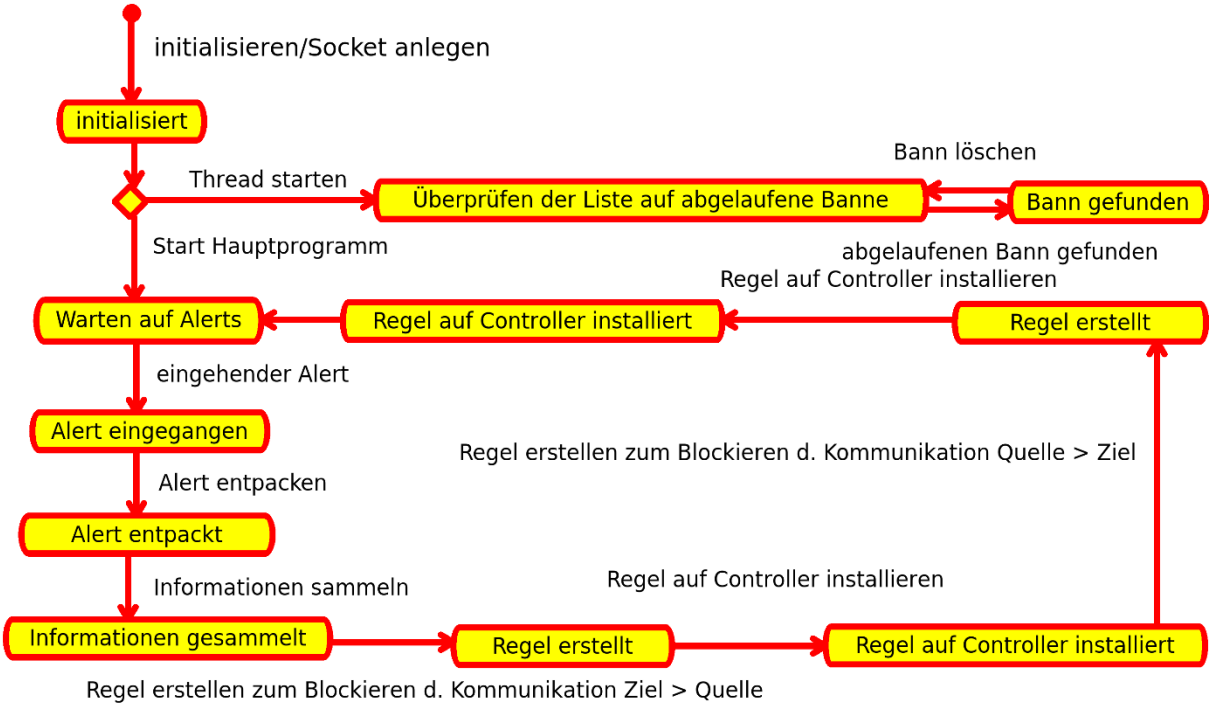

Abb. 2: Zustandsdiagramm

Das Programm nutzte IP-Adressen als Grundlage für die Erstellung der Flows. Würden MAC-Adressen genutzt werden, wäre die Quell-MAC-Adresse eines Angriffes die der letzten Station im Netzwerk. Dadurch wäre anschließend der komplette Datenverkehr zwischen angegriffener Maschine und beispielsweise einem Gateway unterbrochen und auch ungefährlicher Traffic würde blockiert werden.

5 Tests

Um zu zeigen, dass das System wie gewünscht funktioniert, wurden Tests durchgeführt.

Der erste Test zielte auf die Erkennung und Sperrung von einfachen ICMP-Paketen. Diese lassen sich einfach durch einen Ping von einer Maschine zur anderen erzeugen. Dazu musste Snort für die Erkennung von ICMP-Paketen mit der Regel alert ICMP any any <> $HOME_NET any (msg: "ICMP alert") konfiguriert werden. Durch einen TCP-Dump am IDS wurden die am Switch duplizierten ICMP-Pakete sichtbar. Dies zeigte, dass die Paket-Duplizierung am Switch funktionierte. Auch in den Logs des IDS waren die Alerts zu den erkannten ICMP-Paketen

sichtbar. Ob Alerts über den Socket korrekt ausgegeben wurden, konnte durch Start des Programms festgestellt werden. Sofort wurden Alerts empfangen, Flows generiert und an den Controller gesendet. Die Kommandozeilenausgabe zeigte die korrekte Installation auf dem Switch (siehe Abbildung 3).

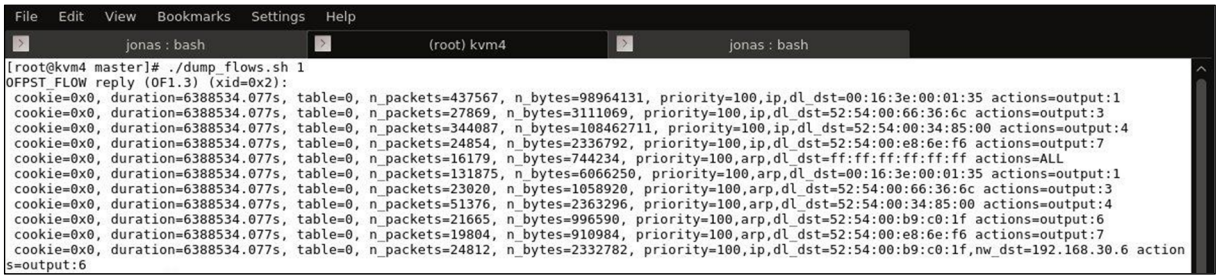

Abb. 3: Am Switch installierte Flows

Dies zeigte, dass die Verknüpfung von einem IDS auf Basis von Snort und OpenDaylight durchaus funktioniert.

In einem weiteren Test sollte untersucht werden, ob auch echte Angriffe blockiert werden können. Als Beispiel sollte ein Portscan dienen. Auf der Zielmaschine wurde Port 22 (SSH) geöffnet und der Scan ausgeführt. Ohne Sperrung am Switch war der Scan erfolgreich und erkannte den Port. Sobald das Programm aktiv war und den Alert empfing wurde der Scan zunächst blockiert (siehe Abbildung 4).

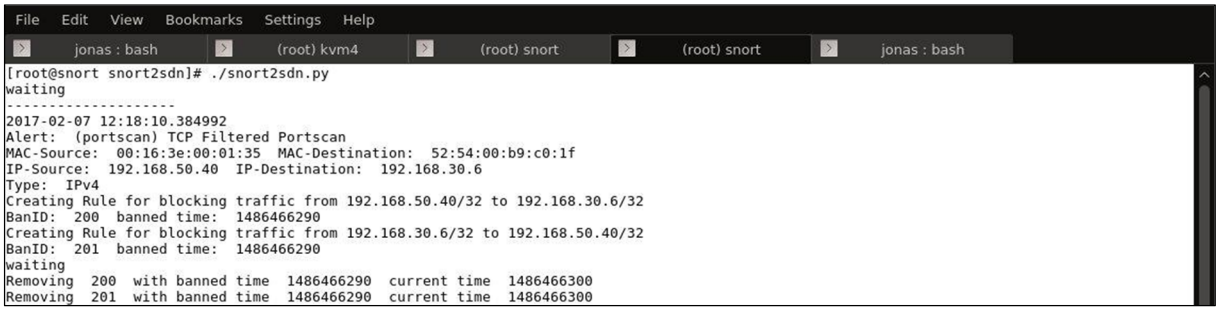

Abb. 4: Vom Programm gesperrter Portscan

Wurde der Flow durch das Programm nach Ablauf der definierten Zeit wieder gelöscht, lief der Scan weiter und war schließlich erfolgreich, allerdings mit starker zeitlicher Verzögerung. Der Grund dafür lag in der Art und Weise, wie das genutzte Tool nmap, arbeitet. So werden zunächst die am häufigsten genutzten Ports in einer zufälligen Reihenfolge mehrfach untersucht. Zudem gab es eine kurze Zeitverzögerung, bis Snort den Scan erkannte, da erst Daten gesammelt werden müssen, um von einem Portscan ausgehen zu können. Diese Verzögerung genügte, um den Scan erfolgreich abzuschließen. Erst als das Zeitintervall groß genug war, schlug der Scan fehl.

Mit den Tests wurden folgende Komponente erfolgreich auf Funktion geprüft:

- Monitor-Port von Open vSwitch
- Snort
- Programm zur Verknüpfung von Snort und OpenDaylight
- Zusammenspiel Switchsoftware Open vSwitch mit Controller OpenDaylight,
- Funktion der Flows zur Blockierung von Traffic

Weiterhin fiel auf, dass ein Neustart des Hostsystems oder einer VM weitreichende Rekonfiguration des Systems mit sich zog. Switchports, über welche die VMs an den Switch angeschlossen sind, und der interne Name des Switches in OpenDaylight sind nicht persistent und werden bei Änderungen neu vergeben. Das bedeutet, dass die entsprechenden Konfigurationen des Portmonitorings in Skripten, XML-Dokumenten der Flows für den Netzwerkverkehr und im Programm angepasst werden müssen.

6 Fazit

Grundsätzlich war es möglich, das Netzwerk auf SDN umzustellen und bestehende Komponenten, hier das Intrusion Detection System Snort, mit OpenFlow zu verknüpfen, um Angriffe auf das Intranet am Switch abwehren zu können. Da keine direkten Schnittstellen zwischen den Modulen verfügbar waren, musste ein eigenes Programm entwickelt werden, welche die einzelnen Teile miteinander verknüpft. Die anschließenden Tests haben gezeigt, dass das entwickelte System funktioniert und Angriffe abwehren kann.

Literatur

[Ere15] E. Eren: „SDN mit TLS absichern“; NET (Zeitschrift für Kommunikationsmanagement), ISSN 0947-4765, 10/2015

[FeRZ13] N. Feamster, J. Rexford, E. Zegura, The Road to SDN, acmqueue, 2013, https://queue.acm.org/detail.cfm?id=2560327

[Fou12] Open Networking Foundation. Software-defined networking: The new norm for networks. Technical report, April 2012

[Fou15a] Open Networking Foundation. Open Networking Foundation. https://www.opennetworking.org/, 2015. abgerufen am 18-März-2017

[Fou15b] Open Networking Foundation. Member Listing https://www.opennetworking.org/our-members, 2015. abgerufen am 18-März-2017